A useful tool: regular expressions

Usage:

Validate, search or replace text (in DW: filter&clean data).

Appear in:

- PHP, javascript (validate input, reformat), SQL (pattern matching)
- script languages/UNIX scriptsp: `grep`, `sed`, `awk`...
- perl
- programming language libraries: `perl`, `python`, `java`, `c++`...
- parsers, packet analysis...
- text editors/IDE (`Find&Replace`))

Caveat:

Several "standards", features vary slightly. Main flavours:

- POSIX flavours: Basic and Extended (BRE, ERE)
- PCRE (originally from Perl).

Regex Engines:

PCRE, Oniguruma, RE2 (Google), Boost (C++), RegExp (Javascript)...

regular expressions: memento:

a	symbol a
`(r)`	`r` (delimiter/capture)
$\mathtt{r}_1 \mid \mathtt{r}_2$	r_1 or r_2 (alternative)
$\mathtt{r}_1\,\mathtt{r}_2$	concatenation

Special characters:

`.`	any symbol
`^`	text beginning
`$`	end of text

Quantification:

$\mathtt{r}?$	0 or 1 occurrence of `r`
$\mathtt{r}*$	0 or more occurrences
$\mathtt{r}+$	1 or more occurrences
$\mathtt{r}\{n\}$	exactly `n` occ.
$\mathtt{r}\{n,\}$	at least `n` occ.
$\mathtt{r}\{min, max\}$	between `min` and `max` occ.

Captured subexpression:

`\n` the substring matching n^{th} captured group

(defined by n^{th} opening parenthesis)

Character classes:

`[` $a_1 \ldots a_n$ `]`	1 character: a_1 or a_2 or...
`[a-d]`	a, b, c or d
`[^...]`	any character *but* ...

Predefined character classes:

`[:alpha:]`	[A-Za-z]
`[:alnum:]`	[A-Za-z0-9]
`[:space:]`	[\t\r\n\v\f]=spaces
`[:punct:]`	punctuation
`[:upper:]`	uppercase
`[:print:]`	[\x20-\x7E]=visible char+space

Predefined character classes (PCRE):

`\d`	decimal
`\h`	horizontal space character
`\s`	vertical space character
`\w`	word item

(uppercase to negate: `\D` = non-number.)

Metacharacters:

`^ . [ ] $ ( ) * + ? | { } \` escaped by \

Metacharacters for PCRE only:

`! < > = :`

regular expressions: examples

Special rules:

- by default, engine searches *first and longest* occurrrence
- POSIX charact classes used within "[]": ex: [[:alpha:]]
- predefined char classes determined by LC_CTYPE category in UNIX *locale*
- beware of digraphs, *é* may be 1 or 2 character, etc.
- *meta* status generally lost inside char class.
- when - first or last in "[]" : no interval but symbol - itself.
- \0 captures whole string
- sometimes $n (outside pattern) instead \n (within) for backreference
- script/prog languages interpret pattern before forwarding to engine ⇒ escape symbols ex: ( ) \ ⇒ double escape!
- . . . many options, vary a lot between tools.

Examples:

- [a-z]+0 matches text containing one or more lowercase followed by 0.
- ^[0-9]{10}$ matches text (line) that is a 10-digit number.
- ([a-c])z\1\1 matches *azaa* but not *azcc*.

regular expressions: memento (advanced):

Misc.:

`[=a=]` equivalence class of "a" [aàáâäåãAÀ...]
defined in `LC_COLLATE` category of UNIX *locale*.

`?:` non-capturing group

Assertions : ! if negative, = if positive, < for lookbehind

`(?=r)` positive lookahead
`(?!r)` negative lookahead
`(?<=r)` positive lookbehind
`(?<!r)` negative lookbehind

Conditional pattern (only in some engines: python, perl, pcre)

`(?(if)then|else)`

Options (non-standard, but generally available under some form)

`i` case-*i*nsensitive
`m` *m*ultiline: if text has symbols, `^` `$` match line extremities
`s` *s*ingle-line: enable "." to match newline char
`x` e*x*panded: spaces ignored unless escaped...

Examples:

- `new(?!s)` over "`Those news seem newer than new`"
- `(?ms)^a(.)*z$` over "`abcd\ngfz`\na"
- regexp to validate password ($\geq$ 8 symbols, digit, punctuation, uppercase) ?

Greedy, lazy/reluctant, possessive quantifiers:

By default quantifiers are *greedy*: from a position, match as many occurrences as possible, then backtrack if no solution for global pattern.

- With `?` quantifier becomes *lazy*: the fewest occurrences, then increases if no solution.
- With `+` quantifier becomes *possessive* (`Java, Python, Perl...`): max occurrences, no backtracking even if it fails.

Examples:

- `ba*` over "a*baaa*c"
- `ba*ac` over "a*baaac*"
- `ba*?` over "a*ba*aac"
- `ba*?c` over "a*baaac*"
- `(ab{2,}+[a-z]` over "a*abbc*"
- `(ab{2,}+[a-z]` over "aabbb"
- `([a-c])*+cz` never matches.

Regular expressions: UNIX, python

- `egrep = grep -E` searches input file, returns lines where pattern found.
  `egrep 'ion$'/usr/dict/words` ... returns words ending in *ion*.
- `sed` very powerful tool. Can modify text.
  `sed -e 's/before/after/g' infile.txt > outfile.txt`
  ... replaces each occ. of before with after.

  ```
  sed -i.back
  's#\([0-9]\{4\}\)-\([0-9]\{2\}\)-\([0-9]\{2\}\)#\3/\2/\1#g' a.txt
  ```
  ... changes date format inplace, backup (we escaped ( )).
- `python`: functions `match`, `search`, `findall`, `sub`.

```
import re
c = re.search('(\d\d?) \w+ \d{4}', 'le  16 avril 2017')
print c.group(1) # 16

pattern = re.compile('\d\d? \w+ \d{4}')
c = pattern.search('le  16 avril 2017')
print c.group()  # '16 avril 2017'

re.sub('([0-9]{4})-([0-9]{2})-([0-9]{2})','\\1/\\2/\\3','2016-04-16')
# '2016/04/16'
```

Regular expressions: SQL

- `Oracle` : POSIX ERE compliant.

```
UPDATE countries
  SET name =  REGEXP_REPLACE(name,'(.)','\1 ') WHERE name != France;
-- name becomes: B r a z i l
SELECT first_name, last_name FROM employees
  WHERE REGEXP_LIKE (first_name, '^Ste(v|ph)en$')
```

- `PostgreSQL` : implements regexp-like patterns with `SIMILAR TO`, some regexp POSIX functions:

```
SELECT col  FROM t WHERE (col similar to '%(b|d)%');
-- returns "abc", but not "aca"
SELECT regexp_replace('foobarbaz', 'b..', 'X', 'g')
--  fooXX
```

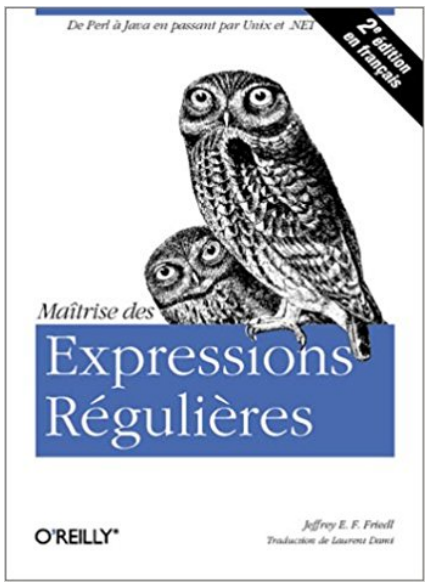
- `MySQL` : Herbert Spencer's `regex` library (POSIX)
- `MariaDB` : PCRE library (prev versions: `regex`)
- `Microsoft SQL Server` : partial support with `LIKE` (afaik)
- `DB2` : no direct support (afaik) $\Rightarrow$ UDF.

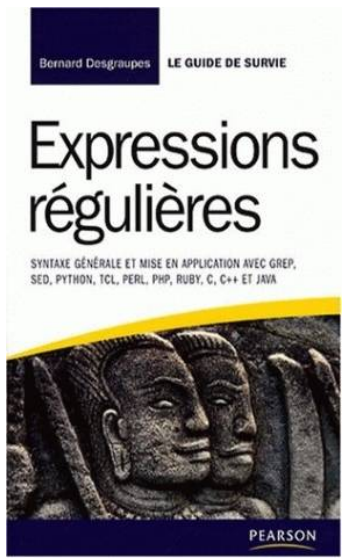
Can also call regexp library through UDF.

Bibliography:

- https://en.wikipedia.org/wiki/Regular_expression
- http://www.regular-expressions.info
- http://www.expreg.com/presentation.php (très complet)
- https://openclassrooms.com/courses/concevez-votre-site-web-avec-php-et-mysql/les-expressions-regulieres-partie-1-2
- https://stackoverflow.com/questions/22937618/reference-what-does-this-regex-mean
- https://regex101.com (tester/débuger une regex)

À la B.U. (Paris-Sud) :





`http://www.regular-expressions.info/books.html` includes some book reviews.